CSE-276C HW4

Zhenyu Wu | PID: A69030822

November 26, 2024

1 Question 1

My general thoughts to approach this question is

1. Apply PCA on the Training data to get the W_{pca} as well as the mean of training data for each image
2. Apply the LDA on the projected training data from PCA $W_{pca}^T X_{train}$ to get the W_{lda}
3. Project training data to the LDA Plane $W_{lda}^T(W_{pca}^T X_{train})$
4. Normalize test data with the mean of training data, project it to the PCA plane $W_{pca}^T X_{test}$, then project it to the LDA plane $W_{lda}^T(W_{pca}^T X_{test})$
5. Fit processed training $W_{lda}^T(W_{pca}^T X_{train})$ and test $W_{lda}^T(W_{pca}^T X_{test})$ to the classifier with classID, and predict on test data.

I followed the lecutre notes as well as the tutorial of OpenCV to implement PCA and LDA at `https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html`. **Once again, I computed W_{pca} and W_{lda} from the training data. I directly use them to project test data**

1.1 Reprocessing

Before PCA and LDA, the data were preprocessed by:

1. Only read the ROI region of interest for each image.
2. Scale each image to 64x64, since, based on my observation, most of them would have h around 40 and w around 50 (for the resize, I computed the ratio between target width and current width as well as height, so the resized image should retain the original aspect ratio of the input image).

After Prepossessing, the image looks like:

Figure 1: Preprocessed Image: Rescale

1.2 PCA

My PCA implementation follows below process:
Given the image data $X = [x_1, x_2, x_3 \dots x_n]$, X is a matrix $m \times n$ where m is the number of features(pixels) of each image, n is the sample size.

1 Compute the mean of training data

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i$$

2 Compute the covariance matrix

$$S = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)(x_i - \mu)^\top$$

I didn't apply the inner product to compute S since each image is just 64×64, which didn't give a massive amount of pixels.

3 Compute the eigenvalue and eigenvector

$$S\mathbf{v}_i = \lambda_i \mathbf{v}_i, \quad i = 1, 2, \dots, n$$

4 Select k^{th} largest eigenvalue and eigenvector to form W_{pca}

5 Calculate Principal Component and reduce dimensionality by projection

$$X_{pca} = W_{pca}^T \cdot X$$

In addition, I also tested my implementation of PCA by successfully reconstructing the original image with $k = 64 \times 64$ following:

$$X = W_{pca} X_{pca} + \mu$$

To choose a reasonable k, I plotted the eigenvalue as below

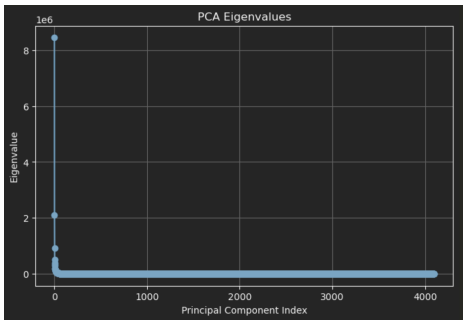

Figure 2: PCA: Eigenvalue

From the plot, the eigenvalue was quite high in range $[0, 1000]$, then I further zoomed in to this range and tested some k value. For example, below are the reconstructed images with $k = 800$ and $k = 400$

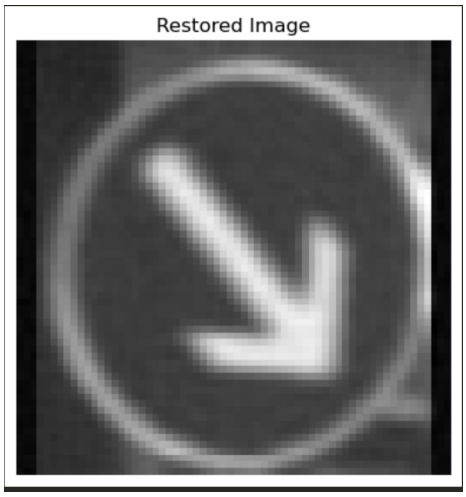

Figure 3: PCA: Reconstructed Image K=800

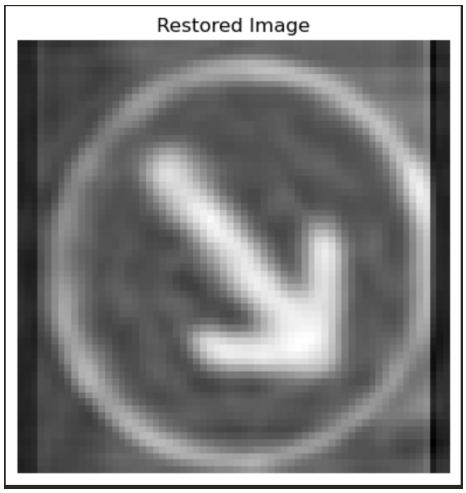

Figure 4: PCA: Reconstructed Image K=400

I tried multiple different k value and find a optimal one $\boldsymbol{k_{opt} = 480}$ based on the accuracy of the classifier. With $k = 480$, I visualized the 1st and 2nd eigenvectors as

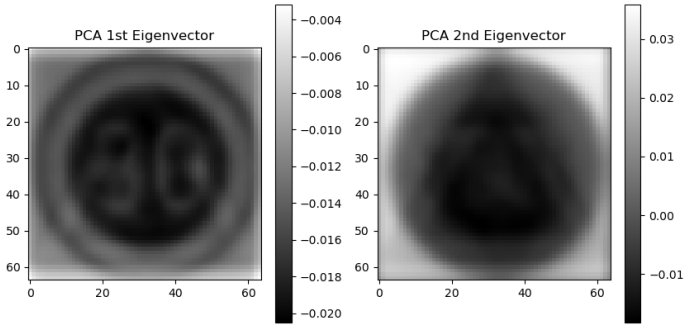

Figure 5: PCA: 1st and 2nd eigenvectors

The W_{pca} has a shape of $(4096, 480)$, each eigenvector in W_{pca} represents a principal component direction. The 1st eigenvector represents the direction of maximum variance in the original data. Based on the visualization, it could present some pixel regions that contribute most to the variance in the dataset, such as the number at the center. For the 2nd eigenvector, it represents the second most significant direction of variance in the data. The visualization might reflect some secondary patterns, such as the triangular shape.

1.3 LDA

My implementation of LDA follows below process:
Like I mentioned, the input data for my LDA are the principal component of original data $X_{pca} = Wpca^T \cdot X$

1 Compute total mean

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i$$

2 Compute mean of each class

$$\mu_i = \frac{1}{|X_i|} \sum_{x_j \in X_i} x_j$$

3 Compute scatter matrices S_W and S_B

$$S_B = \sum_{i=1}^{c} N_i (\mu_i - \mu)(\mu_i - \mu)^\top$$

$$S_W = \sum_{i=1}^{c} \sum_{x_j \in X_i} (x_j - \mu_i)(x_j - \mu_i)^\top$$

4 Solve eigenvalue and eigenvector from

$$S_W^{-1} S_B \mathbf{v}_i = \lambda_i \mathbf{v}_i$$

5 Select $c - 1$ largest eigenvalue and eigenvector to form W_{lda}

6 Make projection to LDA plane

$$X_{lda} = W_{lda} \cdot X_{pca}$$

For the 1st and 2nd eigenvectors, I visualize them as follow through back project them to pixel space:

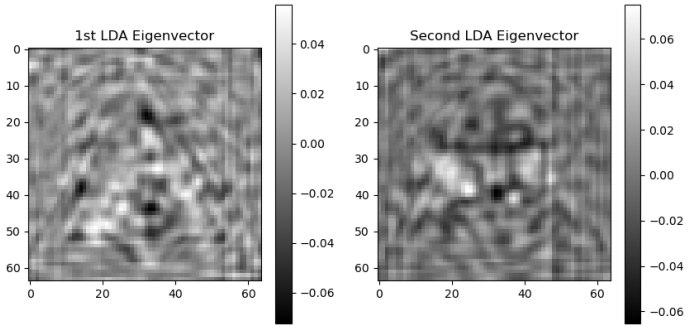

Figure 6: LDA: 1st and 2nd eigenvectors

The largest two LDA eigenvectors represent the two most discriminative directions in the PCA-reduced feature space. The largest eigenvector represents the direction where the between-class variance is maximized relative to the within-class variance. The 2nd eigenvector represents the second-best direction for class separability, orthogonal to the first eigenvector. It captures additional discriminative information that the first eigenvector does not explain.

1.4 Classification

To do classification, I applied following steps:

1 Apply W_{pca}, W_{lda} on training data to get projected $X_{lda-training} = W_{lda}^T(W_{pca}^T X_{train})$

2 Normalize X_{test} with mean of training data, apply W_{pca}, W_{lda} on test data to get projected $X_{lda-test} = W_{lda}^T(W_{pca}^T X_{test})$

3 Fit them to the classifier and record accuracy.

For the classifier, I tried KNN as well as SVM. **My best result was with KNN, where the $k = 480$ for PCA. The accuracy was 0.89944576405384**, which means 89.9% of teat data was correctly classified (**Can find this result in my code)**. I also tried different parameters, especially the k for PCA, but so far I haven't observe any other value could have the better classification rate. For the SVM, it had worse result than KNN.
Based on this result where KNN outperformed SVM, I would say the implementation of PCA and LDA was quite successful. Since KNN relies heavily on the distance between data points. If PCA and LDA created a feature space where samples from different classes are well-separated, KNN will naturally perform better because it can classify data points based on proximity without overfitting or underfitting. This means that PCA and LDA processing likely produced

a feature space where the distances between points are meaningful and reflect class separability.
In addition, I also tested the case where LDA was not applied, the process of this is

1 Apply W_{pca} on training data to get projected $X_{pca-training} = W_{pca}^T X_{train}$

2 Normalize X_{test} with mean of training data, apply W_{pca} on test data to get projected $X_{pca-test} = W_{pca}^T X_{test}$

3 Fit them to the classifier and record accuracy.

For this comparison test PCA + KNN, I kept all parameters the same, as the one for PCA + LDA + KNN. **Without LDA, the accuracy of classification from KNN is 0.47410926365795725** which was significantly lower than the one with LDA. This decline is quite resonable and expected. Like I mentioned, the KNN relies heavily on the distance between data points. Without LDA, we couldn't project data into feature space where the separation between classes are maximized, which significantly impacted the performance of KNN.

2 Question 2

For this question, the general workflow is quite straightforward. **I applied x to represent x_1 which is the prey, and y to represent x_2 the predator. So $x = x_1$ and $y = x_2$**

1 Generate a list of time T from 0 to 30. Initialize the step size $h = \Delta t$

2 Iterate until $T = 30$. For each iteration, compute x, y based on step size Δt with 4-th order Runge-Kutta.

For the prey, $x = x_1$, we have

$$f_x = x - (x \times y)$$

$$\begin{aligned}
k_{1x} &= \Delta t \cdot f_x((x_0, y_0) \\
k_{2x} &= \Delta t \cdot f_x \left(x_0 + \frac{k_{1x}}{2}, y_0 + \frac{1}{2}\Delta t \right) \\
k_{3x} &= \Delta t \cdot f_x \left(x_0 + \frac{k_{2x}}{2}, y_0 + \frac{1}{2}\Delta t \right) \\
k_{4x} &= \Delta t \cdot f_x \left(x_0 + k_{3x}, y_0 + \Delta t \right) \\
x &= x_0 + \frac{1}{6}k_{1x} + \frac{1}{3}k_{2x} + \frac{1}{3}k_{3x} + \frac{1}{6}k_{4x}
\end{aligned}$$

For the predator, $y = x_2$, we have

$$f_y = (x \times y) - y$$

$$
\begin{aligned}
k_{1y} &= \Delta t \cdot f_y(x_0, y_0) \\
k_{2y} &= \Delta t \cdot f_y\left(x_0 + \frac{1}{2}\Delta t, y_0 + \frac{k_{1y}}{2}\right) \\
k_{3y} &= \Delta t \cdot f_y\left(x_0 + \frac{1}{2}\Delta t, y_0 + \frac{k_{2y}}{2}\right) \\
k_{4y} &= \Delta t \cdot f_y\left(x_0 + \Delta t, y_0 + k_{3y}\right) \\
y &= y_0 + \frac{1}{6}k_{1y} + \frac{1}{3}k_{2y} + \frac{1}{3}k_{3y} + \frac{1}{6}k_{4y}
\end{aligned}
$$

As the result, I have

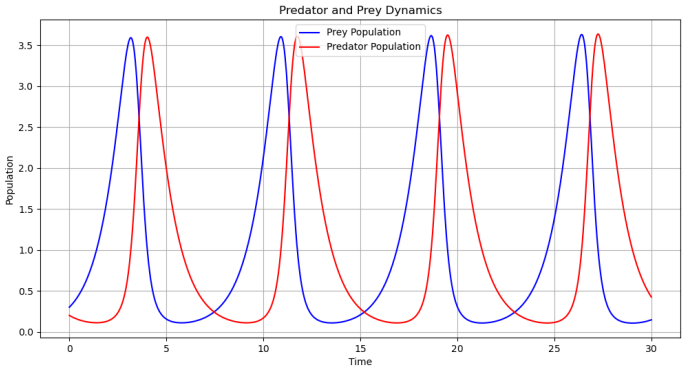

Figure 7: Dynamic